Express Mail No. EL887267562US

IBM DOCKET: ROC920010252US1
WHE DOCKET: IBM-209

# APPLICATION

# FOR

# UNITED STATES LETTERS PATENT

**TITLE:** YIELD ON MULTITHREADED PROCESSORS

**APPLICANT(S):** William Joseph Armstrong, Chris Francois, Naresh Nayar

**ASSIGNEE:** INTERNATIONAL BUSINESS MACHINES CORPORATION

Wood, Herron & Evans, L.L.P.
2700 Carew Tower
Cincinnati, Ohio 45202
513-241-2324

SPECIFICATION

# Yield on Multithreaded Processors

## Field of the Invention

The present invention relates to computing systems, and more particularly, to yielding a CPU within a logically-partitioned environment.

## Background of the Invention

The speed and efficiency of many computing applications depends upon the availability of processing resources. To this end, computing architectures such as the "virtual machine" design, developed by International Business Machines Corporation, share common processing resources among multiple processes. Such an architecture may conventionally rely upon a single computing machine having one or more physical controllers, or central processing units (CPUs). The CPUs may execute software configured to simulate multiple virtual processors. Each virtual processor may embody an independent unit of execution, or thread.

A CPU that can concurrently maintain more than one such unit or path of execution is a called a multithreaded CPU or processor. Each path of execution is

called a thread. In a multithreaded CPU system, each thread performs a specific task that may be executed independently of the other threads. For efficiency purposes, each thread may share some physical resources of a CPU, such as buffers, hardware registers and address translation tables. This hardware architecture mandates that all threads of a multithreaded CPU execute within the same virtual address space. For instance, if a CPU supports two threads, both threads must execute within the same partition or hypervisor, as discussed below.

A partition may logically comprise a portion of a machine's CPUs, memory and other resources, as assigned by an administrator. As such, the administrator may share physical resources between partitions. Each partition will host an operating system and may have multiple virtual processors. In this manner, each partition operates largely as if it is a separate computer. An underlying program called a "hypervisor," or partition manager, may use this scheme to assign and dispatch physical resources to each partition. For instance, the hypervisor may intercept requests for resources from operating systems to globally share and allocate them. If the partitions are sharing processors, the hypervisor allocates physical CPUs between the virtual processors of the partitions sharing the processor.

In an effort to increase the speed of conventional (non-multithreaded), partitioned environments where partitions are sharing processors, system designers commonly implement yield calls. Yield calls generally represent programable attempts to efficiently distribute CPUs among

partitions sharing processing availability. For instance, an operating system executing a thread may issue a yield call to a hypervisor whenever the thread spins in a lock or executes its idle loop. Such an idle thread may have no work to perform, while a locked thread may "spin" as it waits for the holder of the lock to relinquish it. In response to the yield call, the thread may enter an idle state, while the hypervisor reallocates the CPU.

More particularly, a virtual processor that is spinning on a lock held by another virtual processor may initiate a yield-to-active call. In response to the yield-to-active command, the virtual processor may enter an idled state and relinquish its CPU. The hypervisor may reallocate the yielded CPU to the next virtual processor presented on a dispatch schedule of the hypervisor.

Should a thread be in an idle loop, the operating system executing the thread may make a timed-yield. Such a yield call may cause the operating system to relinquish its CPU for a period specified within the yield call. The duration may correspond to an interval of time where the operating system running the thread does not require the CPU that has been dispatched to it. As such, the timed-yield allows the CPU to be utilized by another virtual processor until a time-out event registers. Of note, the virtual processor may be in a different partition. The time-out may coincide with the expiration of the specified interval, at which time the hypervisor will end the yield operation and dispatch a CPU back to the operating system that originally executed the thread.

While such yield applications may succeed in improving the efficiency of some processing systems, known yield processes are not designed for multithreaded CPU environments. Subsequently, yield processes often do not conform to the operating rules and hardware requirements specific to the multithreaded CPU environments. Namely, known yield processes fail to address the requirement that all thread executing on a multithreaded CPU must execute within the same virtual space. Furthermore, conventional yield processes do not regard the independent execution of such threads, nor do they offer a means of monitoring and coordinating thread execution. Consequently, there is a need for an improved manner of managing the allocation of physical computing resources within a multithreaded CPU environment.

Summary of the Invention

One embodiment consistent with the principles of the present invention includes an apparatus, method, and program product configured to facilitate the sharing of physical resources on a multithreaded CPU. More specifically, to coordinate the yielding of multiple threads executing on a multithreaded CPU. The yield of a first thread may be deferred while it waits for at least a second thread of the CPU to become ready to yield. For instance, the embodiment may spin the first thread as it waits for other threads of the CPU to make yield calls. In response to the second thread becoming ready to yield, the first thread may yield, itself. More particularly, the embodiment may place at least the first and second threads of the processor on an idle loop.

A program of the embodiment may additionally save the states of the operating system(s) executing the threads. Alternatively, a program of the embodiment may abandon the yield of the first thread while spinning and after detecting an event, such as a time-out or external I/O interrupt, that is related to the reason that initially required the yield in the first place.

The above and other objects and advantages of the present invention shall be made apparent from the accompanying drawings and the description thereof.

Brief Description of the Drawing

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate embodiments of the invention and, together with a general description of the invention given above, and the detailed description of the embodiments given below, serve to explain the principles of the invention.

Fig. 1 is a block diagram of a computer consistent with the invention;

Fig. 2 is a block diagram of the primary software components and resources in the computer of Fig. 1;

Fig. 3 is a flow chart embodying one method of coordinating yields within the multithreaded CPU environments of Figs. 1 and 2.

Detailed Description of Specific Embodiments

An embodiment of the present invention may include an apparatus, program product and method for accommodating conventional yield calls within a multithreaded CPU environment by coordinating yield processes within the hypervisor. More particularly, a thread encountering a spin lock or idle loop makes a yield call to the hypervisor. In response, the hypervisor gathers and spins yielding threads within the hypervisor to ensure all threads yield within the same virtual space. Namely, the hypervisor spins threads in a ready-to-yield state, until all threads are prepared to yield together. Upon recognizing that all threads are in the requisite ready state, the hypervisor may save the operating system states of all threads and place them in the hypervisor idle loop. Alternatively, a spinning thread may abort its yield call in response to detecting an external I/O interrupt or time-out event. An environment suited for execution of such an embodiment is illustrated in Figs. 1 and 2.

Hardware and Software Environment

Turning to the Drawings, wherein like numbers denote like parts throughout the several views, Fig. 1 illustrates a data processing apparatus or computer 10 consistent with the invention. Apparatus 10 generically represents, for example, any of a number of multi-user computer systems such as a network server, a midrange computer, a mainframe computer, etc. However, it should be appreciated that the invention may be implemented in other data processing apparatus, e.g., in stand-alone or single-user computer systems such as workstations, desktop computers, portable

computers, and the like, or in other computing devices such as embedded controllers and the like. One suitable implementation of apparatus 10 is in a midrange computer such as the AS/400 or e series computer available from International Business Machines Corporation.

Apparatus 10 generally includes one or more multithreaded CPUs 12, or processors, coupled to a memory subsystem including main storage 14, e.g., an array of dynamic random access memory (DRAM). Also illustrated as interposed between multithreaded CPUs 12 and main storage 14 is a cache subsystem 16, typically including one or more levels of data, instruction and/or combination caches, with certain caches either serving individual processors or multiple processors as is well known in the art. Furthermore, main storage 14 is coupled to a number of types of external (I/O) devices via a system bus 18 and a plurality of interface devices, e.g., an input/output bus attachment interface 20, a workstation controller 22 and a storage controller 24, which respectively provide external access to one or more external networks 26, one or more workstations 28, and/or one or more storage devices such as a direct access storage device (DASD) 30.

Fig. 2 illustrates in greater detail the primary software components and resources utilized in implementing a logically partitioned, multithreaded CPU environment on computer 10, including a plurality of logical partitions 40, 42, 44 managed by a partition manager or hypervisor 46. Any number of logical partitions may be supported as is well known in the art.

In the illustrated implementation, logical partition 40 operates as a primary partition, while logical partitions 42 and 44 operate as secondary partitions. A primary partition in this context shares some of the partition management functions for the computer, such as handling the powering on or powering off of the secondary logical partitions on computer 10, or initiating a memory dump of the secondary logical partitions. As such, a portion of hypervisor 46 is illustrated by primary partition control block 50, disposed in the operating system 52 resident in primary partition 40. Other partition management services, which are accessible by all logical partitions, are represented by shared services block 48. However, partition management functionality need not be implemented within any particular logical partition in other implementations consistent with the invention.

Each logical partition utilizes an operating system (e.g., operating systems 52, 54 and 56 for logical partitions 40, 42 and 44, respectively), that controls the primary operations of the logical partition in the same manner as the operating system of a non-partitioned computer. Each logical partition 40-44 executes in a separate memory space, represented by virtual memory 60. Moreover, each logical partition 40-44 is statically and/or dynamically allocated a portion of the available resources in computer 10. For example, each logical partition may share one or more processors 12, as well as a portion of the available memory space for use in virtual memory 60. In this manner, a given processor may be utilized by more than one logical partition.

Additional resources, e.g., mass storage, backup storage, user input, network connections, and the like, are typically allocated to one or more logical partitions in a manner well known in the art. Resources can be allocated in a number of manners, e.g., on a bus-by-bus basis, or on a resource-by-resource basis, with multiple logical partitions sharing resources on the same bus. Some resources may even be allocated to multiple logical partitions at a time. Fig. 2 illustrates, for example, three logical buses 62, 64 and 66, with a plurality of resources on bus 62, including a direct access storage device (DASD) 68, a control panel 70, a tape drive 72 and an optical disk drive 74, allocated to primary logical partition 40. Bus 64, on the other hand, may have resources allocated on a resource-by-resource basis, e.g., with local area network (LAN) adaptor 76, optical disk drive 78 and DASD 80 allocated to secondary logical partition 42, and LAN adaptors 82 and 84 allocated to secondary logical partition 44. Bus 66 may represent, for example, a bus allocated specifically to logical partition 44, such that all resources on the bus, e.g., DASD's 86 and 88, are allocated to the same logical partition.

It will be appreciated that the illustration of specific resources in Fig. 2 is merely exemplary in nature, and that any combination and arrangement of resources may be allocated to any logical partition in the alternative. Moreover, it will be appreciated that in some implementations resources can be reallocated on a dynamic basis to service the needs of other logical partitions. Furthermore, it will be appreciated that resources may also

be represented in terms of the input/output processors (IOP's) used to interface the computer with the specific hardware devices.

The various software components and resources illustrated in Fig. 2 and implementing the embodiments of the invention may be implemented in a number of manners, including using various computer software applications, routines, components, programs, objects, modules, data structures, etc., referred to hereinafter as "computer programs," or simply "programs". The computer programs typically comprise one or more instructions that are resident at various times in various memory and storage devices in the computer, and that, when read and executed by one or more processors in the computer, cause that computer to perform the steps necessary to execute steps or elements embodying the various aspects of the invention.

Moreover, while the invention has and hereinafter will be described in the context of fully functioning computers, those skilled in the art will appreciate that the various embodiments of the invention are capable of being distributed as a program product in a variety of forms, and that the invention applies equally regardless of the particular type of signal bearing medium used to actually carry out the distribution. Examples of signal bearing media include but are not limited to recordable type media such as volatile and non-volatile memory devices, floppy and other removable disks, hard disk drives, magnetic tape, optical disks (e.g., CD-ROM's, DVD's, etc.), among others, and transmission type media such as digital and analog communication links.

In addition, various programs described hereinafter may be identified based upon the application for which they are implemented in a specific embodiment of the invention. However, it should be appreciated that any particular program nomenclature that follows is used merely for convenience, and thus the invention should not be limited to use solely in any specific application identified and/or implied by such nomenclature.

Those skilled in the art will recognize that the exemplary environments illustrated in Figs. 1 and 2 are not intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative hardware and/or software environments may be used without departing from the scope of the invention.

Multithreaded CPU Yield

The flowchart of Fig. 3 illustrates an exemplary embodiment for yielding a CPU within the multithreaded CPU hardware and software environments of the first two figures. Generally, the illustrated process steps account for and coordinate conventional yield requests made by threads executing on a multithreaded CPU by deferring yield until all threads are in a ready-to-yield state. In this manner, the embodiment may ensure that all threads execute within the same virtual address space.

More particularly, a program of the embodiment may cause a thread to spin in response to making a conventional yield call. The thread may continue to spin until the condition prompting the yield call is addressed. For instance, the duration of a thread's timed-yield may expire, or it may receive

an I/O interrupt. Alternatively, the thread may cease spinning in response to recognizing that all other threads of the CPU are likewise in a ready-to-yield state. As such, the yield of all threads are coordinated within the hypervisor, where they may be dispatched to a next occurring virtual processor. In either case, the spinning thread(s) remain within the same partition as all other threads executing on the CPU. As discussed above, such coordination is essential within a multithreaded CPU environment, where threads share common buffers and CPU resources.

Turning particularly to block 169 of Fig. 3, a thread may register that the hypervisor has dispatched a CPU to it. Ideally, such a thread will presently have a job to execute, the processing of which is not predicated upon some interrupt event or lock release. As such, the thread may execute the task at block 169 without wasting CPU cycles. As discussed above, however, such a condition and associated inefficiencies are commonplace within logically-partitioned environments, and often, in fact, prevent the thread from utilizing the allocated CPU.

For instance at block 170, a thread of an operating system may recognize that it is operating within an idle loop. As such, the operating system will determine at block 162 whether the thread has a task to run. If so, the thread will resume execution at block 169. Should the operating system determine that the thread has no work to perform at block 162, then the operating system may make a yield call at block 172 as discussed below. Alternatively at block 163, the operating system may determine that a thread is

spinning on a lock held by another thread. Should the operating system determine at block 163 that the held spin lock is unavailable, then program code of the embodiment may prompt the operating system executing the thread to make a yield call at block 172. As such, a thread makes a yield call in response to either spin lock or idle loop occurrence. In this manner, the embodiment addresses potential inactivity of thread, in that the thread is not actively utilizing its allocated CPU. As discussed above, since the thread holder of the dispatched CPU is inactive, CPU cycles are wasted while the thread is idle or spins.

Of note, the present embodiment is compatible with all conventional yield calls. As discussed below in detail, the embodiment enables known calls within a multithreaded CPU environment by coordinating yield processes within the hypervisor. As discussed above, the type of yield call made by the thread may depend upon the state of the thread. For instance, if a thread is in an idle loop, the operating system executing the thread may make a timed-yield. That is, the yield call made by the operating system to the hypervisor may specify a precise duration. The duration may correspond to an interval of time where the operating system running the thread will not require the CPU dispatched to it. At the expiration of the specified interval, the hypervisor will end the yield operation, and a CPU will be dispatched back to the operating system that originally executed the thread.

Should the inactivity of the thread instead be attributable to a spin lock held by another thread, then the operating system running the thread

may initiate a yield-to-active or yield-to-processor call. The latter call is described in greater detail in U.S. Patent App. S/N _____ filed on even date herewith, by William Joseph Armstrong et al., entitled "System for Yielding to a Processor," the disclosure of which is incorporated by reference. An operating system executing a thread may make a yield-to-active call to the hypervisor whenever the thread is about to enter a spin lock. Conventionally, the yielding thread "spins" as it waits for a prerequisite process to execute or a resource to become available, thereby lifting the lock. In response to the yield-to-active command, the virtual processor may enter an idled state and relinquish its CPU. The hypervisor may reallocate the yielded CPU to the next virtual processor presented on the dispatch schedule of the hypervisor.

Alternatively, a thread caught in a spin lock may initiate a yield-to-processor call. An operating system of such a thread may yield its allocated CPU to a target virtual processor holding the lock. Under such a call, the hypervisor saves the state of the operating system prior to dispatching the surrendered CPU to the operating system of the designated, target virtual processor.

In response to receiving any of the above described yield calls at block 172, the hypervisor may gain control of the thread at block 173. In this manner, the hypervisor now dictates the immediate assignment and actions of the thread previously controlled by the operating system at block 172. Of note, the dashed line bisecting Fig. 3 demarcates the roles performed by the hypervisor and the operating system. More particularly, the operating

system initiates the illustrative steps of blocks 162 - 172, while the hypervisor executes all subsequent blocks. For instance, the hypervisor may cause the thread to spin at block 174. In this manner, all threads of a CPU making a yield call within the multithreaded CPU environment spin within the hypervisor until all threads have similarly made yield calls.

Of note, known yield applications (made outside the context of a multithreaded CPU) conventionally call for a thread to be placed in a hypervisor idle state after making a yield call. The present embodiment departs from conventional yield sequences by instead spinning the thread. This step of block 174 ultimately contributes to coordinating yield processes within a multithreaded CPU environment. Namely, the feature allows all individual threads of a CPU to yield prior to being placed in hypervisor idle loop.

The hypervisor may further mark a shared storage component of the thread making the call at block 177. Of note, such storage may be allocated out of hypervisor storage. More particularly, the hypervisor may mark the storage corresponding to the thread as being "ready-to-yield" at block 173. This ready-to-yield characterization communicates to other threads and to the hypervisor that the thread is prepared to surrender its CPU. Significantly, because the storage is accessible to all other threads of the CPU that have made a yield call, the ready-to-yield designation apprizes these threads of the yielding thread's status. As described below in greater detail,

this shared attribute facilitates yield coordination for each thread of the multithreaded CPU.

While spinning within the hypervisor, the thread may monitor the environment for an event at block 176. More particularly, the thread may continually check to see if the condition for its yield has been satisfied. For instance, a spinning thread may encounter a time-out condition. As discussed above, such an event may correspond to a timed-yield made by a thread in an idle loop at block 172. As such, the time-out may occur at the time proscribed for the yield operation to end. For instance, the operating system originating the timed-yield call reacquires access to CPU cycles.

Another type of event recognizable by the spinning thread at block 176 may embody an external I/O interrupt. Exemplary I/O interrupts may include a disk operation or other external compiling function that may take priority over sequential processing of yielded states. Of note, the interrupt may designate or target the specific operating system of the of the spinning thread.

Should the thread register either a time-out or I/O interrupt event, the hypervisor may return control of the thread back to the operating system at 178. For instance, the operating system running the spinning thread may send a signal to the hypervisor aborting its ready-to-yield call. As such, the yield call may cease at the same instant as the timed-yield ends, or the interrupt occurs, respectively. In this manner, the embodiment affords a spinning thread a final opportunity to realize the event upon which its

execution is predicated, prior to idling. As such, this feature may further accommodate conventional yield calls. Ultimately, a thread responding to an external I/O interrupt or a time-out may return to the operating system at the same point from which it made the yield call. Further, execution of the returned thread continues at block 169 as before the yield call of block 172.

In addition to monitoring the multithreaded CPU environment for time-out and interrupt events at block 176, the spinning thread may also evaluate the states of other threads of the CPU. More particularly, the thread may check the storage of each thread to see if they are, themselves, in a ready-to-yield state. As discussed above, thread status is kept in hypervisor storage and is visible to all threads that enter the hypervisor through a yield call.

Should the hypervisor determine at block 180 that all threads of the CPU are not ready to yield, then the thread will continue to spin within the hypervisor at block 174. As before, the thread will cycle through the repeated monitoring operations of blocks 176 and, if necessary, block 180.

Alternatively, should the hypervisor recognize at block 180 that all threads of a CPU are uniformly in a ready-to-yield state, then each thread may save the state of its corresponding operating system at block 182. Saved states may include applicable registers and thread data. The hypervisor may further store the state in such a manner that the thread becomes active at the same point within the operating system in response to the hypervisor dispatching another virtual processor.

Storing the states as such may prepare the threads to become idle at block 184 of Fig. 3. Subsequently, all threads may enter an idle state within the common virtual space of the hypervisor. In so doing, the embodiment fulfills the above discussed requirement governing multithreaded CPU systems. Namely, all threads execute within the same virtual address space. All yielded threads may further remain idle at block 184 until such time as the hypervisor dispatches another or the same partition to all of the threads. The operating system then regains control of all threads and recalls the states saved at block 182. Of note, the operating system acquires control of all threads at the same point where it originally yielded them to the hypervisor at block 173.

While the present invention has been illustrated by a description of various embodiments and while these embodiments have been described in considerable detail, it is not the intention of the applicants to restrict, or in any way limit, the scope of the appended claims to such detail. For instance, all or part of the coordination of yielding threads in another embodiment may be conducted within individual operating systems or partitions, as opposed to at the hypervisor level. As such, all threads may yield to the hypervisor simultaneously.

Additional advantages and modifications will readily appear to those skilled in the art. The invention in its broader aspects is therefore not limited to the specific details, representative apparatus and method, and illustrative example shown and described. Accordingly, departures may be

made from such details without departing from the spirit or scope of applicant's general inventive concept.

What is claimed is: